Trabajo Guiado MOA

Juanjo Sierra

25 de abril de 2019

Ejercicio 1

Se pide comparar la eficacia de un Hoeffding Tree con un clasificador Naïve Bayes, para un flujo de datos de 1.000.000 de instancias generadas con un generador RandomTreeGenerator, suponiendo una frecuencia de muestreo de 10.000 y con el método de evaluación Interleaved Test-Then-Train.

Aquí vemos como se ejecuta un modelo con Naïve Bayes.

```
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain \
-l bayes.NaiveBayes -s generators.RandomTreeGenerator -i 1000000 -f 10000"
```

Y aquí el ejemplo análogo con Hoeffding Tree.

```
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain \
-l trees.HoeffdingTree -s generators.RandomTreeGenerator -i 1000000 -f 10000"
```

Hay que generar una población de resultados, ejecutando cada uno de estos ejemplos anteriores 30 veces con 30 semillas distintas, y luego mediante un test comparar si existen diferencias significativas en los resultados obtenidos por los dos algoritmos. La generación de las poblaciones de resultados se puede conseguir mediante scripts.

Una vez obtenidos los datos, cada archivo convenientenemente renombrado acorde al algoritmo que lo ha generado, los podemos leer con R.

```
poblacionNaiveBayes = array(dim = 30)
for (i in 1:30) {
  archivo =
    paste(c(paste(c("./Datos/Ejercicio1/naiveBayes",i),collapse = ""),".csv"),collapse="")
  datos = read.csv(archivo)
  accuracyFinal =
    datos$classifications.correct..percent.[length(datos$classifications.correct..percent.)]
  poblacionNaiveBayes[i] = accuracyFinal
}
```

Y a continuación leemos los de Hoeffding.

```
poblacionHoeffding = array(dim = 30)
for (i in 1:30) {
  archivo =
    paste(c(paste(c("./Datos/Ejercicio1/hoeffding",i),collapse = ""),".csv"),collapse="")
  datos = read.csv(archivo)
  accuracyFinal =
    datos$classifications.correct..percent.[length(datos$classifications.correct..percent.)]
  poblacionHoeffding[i] = accuracyFinal
}
```

Para saber si hacer un test paramétrico o no paramétrico para comparar los resultados de ambos algoritmos, primero evaluamos con el test de Shapiro-Wilk si siguen una distribución normal ambas poblaciones.

```r
shapiro.test(poblacionNaiveBayes)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  poblacionNaiveBayes
## W = 0.97381, p-value = 0.6478
```

```r
shapiro.test(poblacionHoeffding)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  poblacionHoeffding
## W = 0.9824, p-value = 0.8852
```

Como según los resultados las dos poblaciones siguen una distribución normal podemos realizar un test paramétrico como el test T de Student.

```r
t.test(poblacionNaiveBayes, poblacionHoeffding)
```

```
##
##  Welch Two Sample t-test
##
## data:  poblacionNaiveBayes and poblacionHoeffding
## t = -1055.4, df = 45.572, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -20.91581 -20.83615
## sample estimates:
## mean of x mean of y
##  73.67307  94.54905
```

En base a los resultados obtenidos podemos afirmar que la diferencia en media de los resultados de los dos algoritmos no es igual a 0, es decir, existen diferencias significativas. En este caso, el algoritmo Hoeffding funciona mejor que el algoritmo Naïve Bayes, pues a igualdad de distribución tiene una media mayor.

Ejercicio 2

Se pide generar 100.000 instancias utilizando el generador de datos SEAGenerator, con un desvío de concepto centrado en la instancia 20.000 en una ventana de 100 instancias. Para simular el desvío de concepto, hacer que el simulador genere la función –f 2 al principio, y luego la función –f 3. Usar un clasificador Naïve Bayes evaluado con una frecuencia de muestreo de 1.000 instancias, usando el método prequential para evaluación. Inserte la configuración directamente en la GUI de MOA para visualizar la gráfica de la evolución de la tasa de aciertos (medida accuracy). ¿Qué se observa?

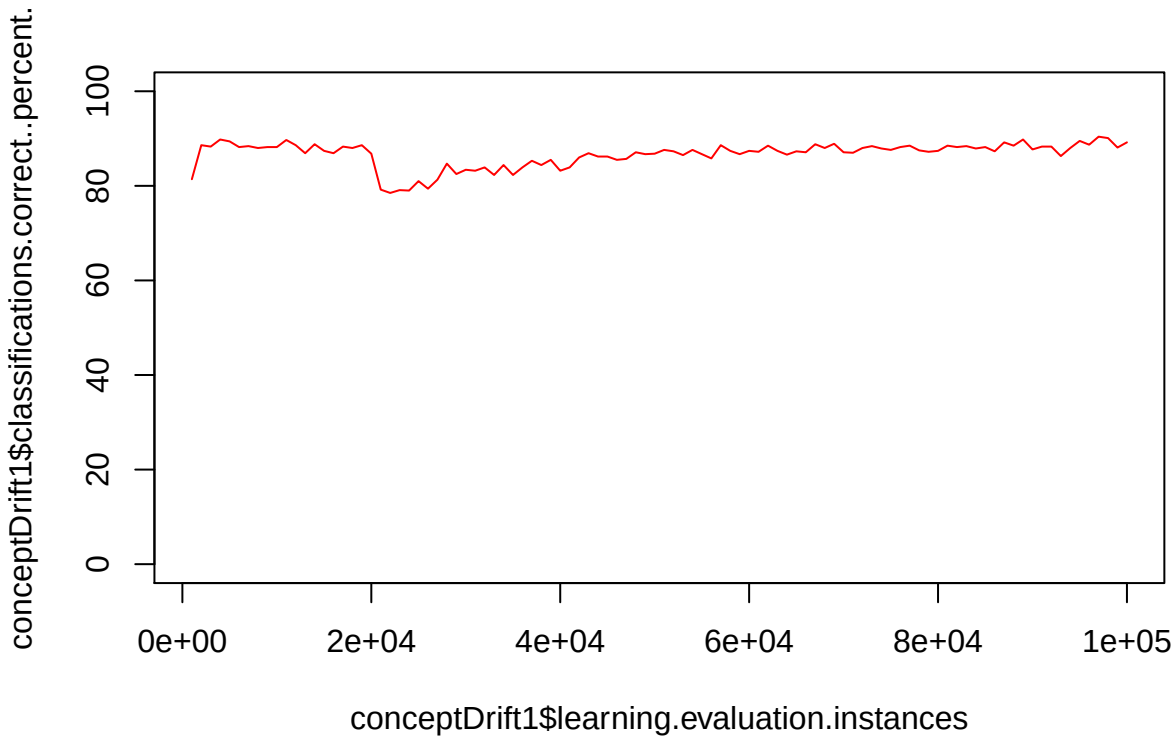
La tarea es evaluar en prequential, sobre el modelo Naïve Bayes, generando 100.000 instancias con frecuencia de muestreo de 1.000.

```
EvaluatePrequential -l bayes.NaiveBayes -s (ConceptDriftStream -s \
(generators.SEAGenerator -f 2) -d (generators.SEAGenerator -f 3) -p \
20000 -w 100) -i 100000 -f 1000
```

Mostramos el valor del accuracy en el tiempo para este clasificador.

```r
conceptDrift1 = read.csv("Datos/Ejercicio2/conceptDrift1.csv")

# Mostrar el acierto del clasificador en el tiempo
plot(conceptDrift1$learning.evaluation.instances,
     conceptDrift1$classifications.correct..percent.,
     "l", ylim = c(0,100), col = "red")
```




```r
# Mostrar el último valor de accuracy y kappa junto a los valores medios
conceptDrift1$classifications.correct..percent.[
  length(conceptDrift1$classifications.correct..percent.)]
```

```
## [1] 89.2
```

```r
mean(conceptDrift1$classifications.correct..percent.)
```

```
## [1] 86.628
```

```r
conceptDrift1$Kappa.Statistic..percent.[
  length(conceptDrift1$Kappa.Statistic..percent.)]
```

```
## [1] 72.84481
```

```r
mean(conceptDrift1$Kappa.Statistic..percent.)
```

```
## [1] 68.61815
```

Se observa que el modelo aprende después del cambio de concepto y se recupera hasta volver a llegar a un nivel aceptable de acierto.

Ejercicio 3

Entrenar un modelo estático Naïve Bayes sobre 100.000 instancias de la función 2 del generador SEAGenerator. Seguidamente, evaluarlo con un flujo de datos con desvío de concepto generado por el generador de datos SEAGenerator, con un desvío de concepto centrado en la instancia 20.000 en una ventana de 100 instancias. Para simular el desvío de concepto, hacer que el simulador genere la función –f 2 al principio, y luego la función –f 3.

Para alcanzar el objetivo pedido hay que ejecutar el siguiente comando:

```
EvaluateModel -m (LearnModel -l bayes.NaiveBayes -s (generators.SEAGenerator -f 2) \
-m 100000) -s (ConceptDriftStream -s (generators.SEAGenerator -f 2) -d \
(generators.SEAGenerator -f 3) -p 20000 -w 100) -i 100000
```

Si comprobamos las soluciones obtenidas (almacenadas en `Datos/Ejercicio3/conceptDrift2.csv`) vemos lo siguiente.

classified instances = 100.000	
classifications correct (percent) = 80	344
Kappa Statistic (percent) = 57	301
Kappa Temporal Statistic (percent) = 54	583
Kappa M Statistic (percent) = 39	98
model training instances = 100.000	
model serialized size (bytes) = 1.840	

En este caso el modelo al ser estacionario aprende previamente sobre unos datos y no sigue aprendiendo en el tiempo, como sí hace el dinámico. Por tanto a partir del cambio de concepto al accuracy del modelo estacionario desciende y nunca la recupera.

Ejercicio 4

¿Qué ocurriría si pudiésemos detectar un cambio de concepto y re-entrenar un modelo estacionario?. El resultado no sería un modelo "estacionario", sino múltiples de ellos entrenados tras detectar cambio de concepto. Se pide: Evaluar, y entrenar online con el método Test-ThenTrain, un modelo estacionario Naïve Bayes que se adapta (re-entrena) tras la detección de un cambio de concepto mediante el método DDM (función SingleClassifierDrift). Usar el flujo de datos del ejercicio anterior.

El comando a utilizar para obtener el resultado que pide el enunciado es el siguiente:

```
EvaluateInterleavedTestThenTrain -l (moa.classifiers.drift.SingleClassifierDrift -l \
bayes.NaiveBayes -d DDM) -s (ConceptDriftStream -s (generators.SEAGenerator -f 2) -d \
(generators.SEAGenerator -f 3) -p 20000 -w 100) -i 100000
```

Esto se ejecuta y se guarda en un fichero csv para su posterior lectura y análisis con R.

```
conceptDrift3 = read.csv("Datos/Ejercicio4/conceptDrift3.csv")

# Mostrar el último valor de accuracy y kappa junto a los valores medios
conceptDrift3$classifications.correct..percent.[length(conceptDrift3$classifications.correct..percent.)]
```

```
## [1] 88.086
```

```r
mean(conceptDrift3$classifications.correct..percent.)
```

```
## [1] 88.086
```

```r
conceptDrift3$Kappa.Statistic..percent.[length(conceptDrift3$Kappa.Statistic..percent.)]
```

```
## [1] 71.23614
```

```r
mean(conceptDrift3$Kappa.Statistic..percent.)
```

```
## [1] 71.23614
```

Como se puede observar, en este caso el modelo sí que está aprendiendo tras el cambio de concepto, y de hecho aprende constantemente. Los resultados son sustancialmente mejores tanto en accuracy como en el estadístico Kappa. Tras un cambio de concepto, un modelo estacionario deja de funcionar, pero un modelo dinámico puede seguir funcionando correctamente.